

A Chomsky-Schützenberger representation for weighted multiple context-free languages*

Tobias Denkinger

Faculty of Computer Science

Technische Universität Dresden

01062 Dresden, Germany

tobias.denkinger@tu-dresden.de

2016-11-29

Abstract

We prove a Chomsky-Schützenberger representation theorem for multiple context-free languages weighted over complete commutative strong bimonoids.

Contents

1	Introduction	2
2	Preliminaries	2
3	Multiple Dyck languages	8
3.1	The original definition	8
3.2	Congruence multiple Dyck languages	9
3.3	Membership in a congruence multiple Dyck language	12
4	CS theorem for weighted MCFLs	15
4.1	Separating the weights	17
4.2	Strengthening the unweighted CS representation	19
4.3	Composing the homomorphisms	21
4.4	The weighted CS representation	22
5	Conclusion and outlook	22

*This is an extended version of a paper with the same title presented at the 12th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP 2015), [Den15].

1 Introduction

Mildly context-sensitive languages receive much attention in the natural language processing community [Kal10]. Many classes of mildly context-sensitive languages are subsumed by the multiple context-free languages, e.g. the languages of head grammars, linear context-free rewriting systems [SMFK91], combinatory categorial grammars [VSWJ86, WJ88], linear indexed grammars [VS87], minimalist grammars, [Mic01b, Mic01a], and finite-copying lexical functional grammars [SNK⁺93].

The Chomsky-Schützenberger (CS) representation for context-free languages [CS63, Proposition 2] has been generalised to a variety of unweighted and weighted settings, e.g. context-free languages weighted with commutative semirings [SS78, Theorem 4.5], tree adjoining languages [Wei88, Lemma 3.5.2], multiple context-free languages [YKS10, Theorem 3], context-free languages weighted with unital valuation monoids [DV13, Theorem 2], yields of simple context-free tree languages [Kan14, Theorem 8.3], indexed languages ([DPS79, Theorems 1 and 2]; [FV15, Theorem 4]; and [FV16, Theorem 18]), and automata with storage weighted with unital valuation monoids [HV15, Theorem 11].

We give a generalisation to the case of multiple context-free languages weighted with a complete commutative strong bimonoid. Sections 3 and 4 contain the main contributions of this paper. The outline of these sections is:

- In order to obtain a CS representation for multiple context-free languages, Yoshinaka, Kaji, and Seki [YKS10] introduce multiple Dyck languages. We give a more algebraic definition of multiple Dyck languages using congruence relations together with a decision algorithm for membership that is strongly related to these congruence relations (Section 3).
- In Section 4 we provide a CS representation for weighted multiple context-free languages by means of a modular proof that first separates the weights from the given grammar and then employs the result for the unweighted case (using the same overall idea as in Droste and Vogler [DV13]).

Since our proofs do not require distributivity, we can be slightly more general than complete commutative semirings. The weight algebras considered here are therefore the complete commutative strong bimonoids.

2 Preliminaries

In this section we recall formalisms used in this paper and fix some notation: We denote by \mathbb{N} the set of natural numbers (including zero). For every $n \in \mathbb{N}$ we abbreviate $\{1, \dots, n\}$ by $[n]$. Let A be a set. The *power set of A* is denoted by $\mathcal{P}(A)$. Let B be a finite set. A *partition of B* is a set $\mathfrak{P} \subseteq \mathcal{P}(B)$ where the elements of \mathfrak{P} are non-empty, pairwise disjoint, and $\bigcup_{p \in \mathfrak{P}} p = B$.

Let A and B be sets and $A' \subseteq A$. The *set of functions from A to B* is denoted by $A \rightarrow B$, we still write $f : A \rightarrow B$ rather than $f \in A \rightarrow B$. Let f and g be functions. The *domain and range of f* are denoted by $\text{dom}(f)$ and $\text{rng}(f)$, respectively. The *restriction*

of f to A' , denoted by $f|_{A'}$, is a function from A' to B such that $f|_{A'}(a') = f(a')$ for every $a' \in A'$. We denote the function obtained by applying g after f by $g \circ f$. Let F be a set of functions and $B \subseteq \bigcap_{f \in F} \text{dom}(f)$. The set $\{f(B) \mid f \in F\} \subseteq \mathcal{P}(\text{rng}(f))$ is denoted by $F(B)$. Let G and H be sets of functions. The set $\{h \circ g \mid h \in H, g \in G\}$ of functions is denoted by $H \circ G$.

Let A be a set and $\approx \subseteq A \times A$ a binary relation on A . We call \approx an *equivalence relation* (on A) if it is reflexive, symmetric, and transitive. Let $a \in A$ and \approx be an equivalence relation. The *equivalence class of a in \approx* , denoted by $[a]_{\approx}$, is $\{b \in A \mid a \approx b\}$. Let $f : A^k \rightarrow A$ be a function. We say that \approx *respects f* if for every $(a_1, b_1), \dots, (a_k, b_k) \in \approx$ holds $f(a_1, \dots, a_k) \approx f(b_1, \dots, b_k)$. Now let \mathcal{A} be an algebra with underlying set A . We call \approx a *congruence relation* (on \mathcal{A}) if \approx is an equivalence relation and respects every operation of \mathcal{A} .

Sorts

We will use the concept of sorts to formalise restrictions on building terms (or trees), e.g. derivation trees or terms over functions. One can think of sorts as data types in a programming language: Every concrete value has a sort (type) and every function requires its arguments to be of fixed sorts (types) and returns a value of some fixed sort (type).

Let S be a countable set (of *sorts*) and $s \in S$. An *S -sorted set* is a tuple (B, sort) where B is a set and sort is a function from B to S . We denote the preimage of s under sort by B_s and abbreviate (B, sort) by B ; sort will always be clear from the context. Let A be an $(S^* \times S)$ -sorted set. The *set of terms over A* , denoted by T_A , is the smallest S -sorted set T where $\xi = a(\xi_1, \dots, \xi_k) \in T_s$ if there are $s, s_1, \dots, s_k \in S$ such that $a \in A_{(s, s_1 \dots s_k)}$ and $\xi_i \in T_{s_i}$ for every $i \in [k]$. Let $\xi = a(\xi_1, \dots, \xi_k) \in T_A$. The *set of positions in ξ* is defined as $\text{pos}(\xi) = \{\varepsilon\} \cup \{iu \mid i \in [k], u \in \text{pos}(\xi_i)\}$ and for every $\pi \in \text{pos}(\xi)$ the *symbol in ξ at position π* is defined as $\xi(\pi) = a$ if $\pi = \varepsilon$ and as $\xi(\pi) = \xi_i(u)$ if $\pi = iu$ for some $i \in [k]$ and $u \in \text{pos}(\xi_i)$.

Weight algebras

A *monoid* is an algebra $(\mathcal{A}, \cdot, 1)$ where \cdot is associative and 1 is neutral with respect to \cdot . A *bimonoid* is an algebra $(\mathcal{A}, +, \cdot, 0, 1)$ where $(\mathcal{A}, +, 0)$ and $(\mathcal{A}, \cdot, 1)$ are monoids. We call a bimonoid *strong* if $(\mathcal{A}, +, 0)$ is commutative and for every $a \in \mathcal{A}$ we have $0 \cdot a = 0 = a \cdot 0$. Intuitively, a strong bimonoid is a semiring without distributivity. A strong bimonoid is called *commutative* if $(\mathcal{A}, \cdot, 1)$ is commutative. A commutative strong bimonoid is *complete* if there is an infinitary sum operation \sum that maps every indexed family of elements of \mathcal{A} to \mathcal{A} , extends $+$, and satisfies infinitary associativity and commutativity laws [DV13, Section 2]:

- (i) $\sum_{i \in \emptyset} a(i) = 0$;
- (ii) for every $j \in I : \sum_{i \in \{j\}} a(i) = a(j)$;
- (iii) for every $j, k \in I$ with $j \neq k : \sum_{i \in \{j, k\}} a(i) = a(j) + a(k)$; and

(iv) for every countable set J and family $\bar{I} : J \rightarrow \mathcal{P}(I)$ with $I = \bigcup_{j \in J} \bar{I}(j)$ and for every $j, j' \in J$ with $j \neq j' \implies \bar{I}(j) \cap \bar{I}(j') = \emptyset$, we have $\sum_{j \in J} \sum_{i \in \bar{I}(j)} a(i) = \sum_{i \in I} a(i)$.

For the rest of this paper let $(\mathcal{A}, +, \cdot, 0, 1)$, abbreviated by \mathcal{A} , be a complete commutative strong bimonoid.

Example 2.1. We provide a list of complete commutative strong bimonoids [DSV10, Example 1] some of which are relevant for natural language processing:

- Any complete commutative semiring, e.g.
 - the *Boolean semiring* $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$,
 - the *probability semiring* $\text{Pr} = (\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$,
 - the *Viterbi semiring* $([0, 1], \max, \cdot, 0, 1)$,
 - the *tropical semiring* $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$,
 - the *arctic semiring* $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$,
- any complete lattice, e.g.
 - any non-empty finite lattice $(L, \vee, \wedge, 0, 1)$ where L is a non-empty finite set,
 - the lattice $(\mathcal{P}(A), \cup, \cap, \emptyset, A)$ where A is an arbitrary set,
 - the lattice $(\mathbb{N}, \text{lcm}, \text{gcd}, 1, 0)$,
- the *tropical bimonoid* $(\mathbb{R}_{\geq 0} \cup \{\infty\}, +, \min, 0, \infty)$,
- the *arctic bimonoid* $(\mathbb{R}_{\geq 0} \cup \{-\infty\}, +, \max, 0, -\infty)$, and
- the algebras $\text{Pr}_1 = ([0, 1], \oplus_1, \cdot, 0, 1)$ and $\text{Pr}_2 = ([0, 1], \oplus_2, \cdot, 0, 1)$ where $a \oplus_1 b = a + b - a \cdot b$ and $a \oplus_2 b = \min\{a + b, 1\}$ for every $a, b \in [0, 1]$.

where \mathbb{R} and $\mathbb{R}_{\geq 0}$ denote the set of reals and the set of non-negative reals, respectively; $+$, \cdot , \max , \min denote the usual operations; \wedge , \vee denote disjunction and conjunction, respectively, for the boolean semiring and join and meet, respectively, for any non-empty finite lattice; and lcm and gcd are binary functions that calculate the least common multiple and the greatest common divisor, respectively.

Also, there are some bimonoids that are interesting for natural language processing but are *not* complete commutative strong bimonoids. E.g.

- the *semiring of formal languages* $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$ where Σ is an alphabet and \cdot is language concatenation, i.e. $L_1 \cdot L_2 = \{uv \mid u \in L_1, v \in L_2\}$ for every $L_1, L_2 \subseteq \Sigma^*$; and
- the semiring $(\Sigma^* \cup \{\infty\}, \wedge, \cdot, \infty, \varepsilon)$ where Σ is an alphabet, \cdot is concatenation, \wedge calculates the longest common prefix of its arguments, and ∞ is a new element that is neutral with respect to \wedge and annihilating with respect to \cdot [Moh00].

Both examples are not commutative. □

An \mathcal{A} -*weighted language* (over Δ) is a function $L : \Delta^* \rightarrow \mathcal{A}$. The *support* of L , denoted by $\text{supp}(L)$, is $\{w \in \Delta^* \mid L(w) \neq 0\}$. If $|\text{supp}(L)| \leq 1$, we call L a *monomial*. We write $\mu.w$ for L if $L(w) = \mu$ and for every $w' \in \Delta^* \setminus \{w\}$ we have $L(w') = 0$.

Recognisable languages

For the many known results concerning finite state automata and regular languages, we will rely on [HU69] and [HU79]. We nevertheless recall the basic definitions:

Definition 2.2. A *finite state automaton*, for short: FSA, is a tuple $\mathcal{M} = (Q, \Delta, q_0, F, T)$ where Δ is an alphabet (*terminals*), Q is a finite set (*states*) disjoint from Δ , $q_0 \in Q$ (*initial state*), $F \subseteq Q$ (*final states*), and $T \subseteq Q \times \Delta^* \times Q$ is a finite set (*transitions*). \square

A *run in \mathcal{M}* is a string $\kappa \in (Q \cup \Delta)^*$ where for every substring of κ of the form quq' (for some $q, q' \in Q$ and $u \in \Delta^*$) we have that $(q, u, q') \in T$, we say that the transition (q, u, q') occurs in κ , the first symbol of κ is q_0 , and the last symbol of κ is in F . The *word corresponding to κ* is obtained by removing the elements of Q from κ . The *language of \mathcal{M}* is denoted by $L(\mathcal{M})$. The *set of recognisable languages*, denoted by REG, is the set of languages L for which there is an FSA \mathcal{M} with $L = L(\mathcal{M})$.

Weighted string homomorphisms

Definition 2.3. Let Δ and Γ be alphabets and $g : \Delta \rightarrow (\Gamma^* \rightarrow \mathcal{A})$ (i.e. a function that takes an element of Δ and returns a function that takes an element of Γ^* and returns an element of \mathcal{A}) such that $g(\delta)$ is a monomial for every $\delta \in \Delta$. We define $\widehat{g} : \Delta^* \rightarrow (\Gamma^* \rightarrow \mathcal{A})$ where for every $k \in \mathbb{N}$, $\delta_1, \dots, \delta_k \in \Delta$, and $u \in \Gamma^*$ we have

$$\widehat{g}(\delta_1 \cdots \delta_k)(u) = \sum_{\substack{u_1, \dots, u_k \in \Gamma^* \\ u = u_1 \cdots u_k}} g(\delta_1)(u_1) \cdot \dots \cdot g(\delta_k)(u_k).$$

We call \widehat{g} an \mathcal{A} -*weighted (string) homomorphism*. \square

Clearly, $\widehat{g}(u)$ is a monomial for every $u \in \Delta^*$. We call \widehat{g} *alphabetic* if there is a function $h : \Delta \rightarrow (\Gamma \cup \{\varepsilon\} \rightarrow \mathcal{A})$ with $\widehat{g} = \widehat{h}$. If $\widehat{g}(u) = \mu.w$ for $u \in \Delta^*$, then we will sometimes say “ \widehat{g} maps u to w ” (leaving out the weight μ) or “ \widehat{g} weights u with μ ” (leaving out the word w). Now assume that $\mathcal{A} = \mathbb{B}$ and we have $|\text{supp}(g(\delta))| = 1$ for every $\delta \in \Delta$. Then g can be construed as a function from Δ to Γ^* and \widehat{g} can be construed as a function from Δ^* to Γ^* . In this case we call \widehat{g} a *(string) homomorphism*. The sets of all \mathcal{A} -weighted homomorphisms, \mathcal{A} -weighted alphabetic homomorphisms, homomorphisms, and alphabetic homomorphisms are denoted by $\text{HOM}(\mathcal{A})$, $\alpha\text{HOM}(\mathcal{A})$, HOM , and αHOM , respectively.

Weighted multiple context-free languages

We fix a set $X = \{x_i^j \mid i, j \in \mathbb{N}_+\}$ of *variables*. Let Δ be an alphabet. The set of *composition representations over Δ* is the $(\mathbb{N}^* \times \mathbb{N})$ -sorted set RF_Δ where for every $s_1, \dots, s_\ell, s \in \mathbb{N}$ we define $X_{(s_1 \dots s_\ell, s)} = \{x_i^j \mid i \in [\ell], j \in [s_i]\} \subseteq X$ and $(\text{RF}_\Sigma)_{(s_1 \dots s_\ell, s)}$ as the set that contains $[u_1, \dots, u_s]_{(s_1 \dots s_\ell, s)}$ for every $u_1, \dots, u_s \in (\Delta \cup X_{(s_1 \dots s_\ell, s)})^*$. We will often write X_f instead of $X_{(s_1 \dots s_\ell, s)}$. Let $f = [u_1, \dots, u_s]_{(s_1 \dots s_\ell, s)} \in \text{RF}_\Sigma$. The *string function of f* , also denoted by f , is the function from $(\Delta^*)^{s_1} \times \dots \times (\Delta^*)^{s_\ell}$

to $(\Delta^*)^s$ such that $f((w_1^1, \dots, w_1^{s_1}), \dots, (w_\ell^1, \dots, w_\ell^{s_\ell})) = (u'_1, \dots, u'_s)$ where (u'_1, \dots, u'_s) is obtained from (u_1, \dots, u_s) by replacing each occurrence of x_i^j by w_i^j for every $i \in [\ell]$ and $j \in [s_\ell]$. The set of all string functions for some composition representation over Δ is denoted by F_Δ . From here on we no longer distinguish between composition representations and string functions. We define the *rank* of f , denoted by $\text{rank}(f)$, and the *fan-out* of f , denoted by $\text{fan-out}(f)$, as ℓ and s , respectively. The string function f is called *linear* if in $u_1 \cdots u_s$ every element of X_f occurs at most once, f is called *non-deleting* if in $u_1 \cdots u_s$ every element of X_f occurs at least once, and f is called *terminal-free* if $u_1, \dots, u_s \in X_f^*$. The subscript is dropped from the string function if its sort is clear from the context.

Note that for every $s' \in \mathbb{N}^* \times \mathbb{N}$, the set of linear terminal-free string functions of sort s' is finite.

Definition 2.4. A *multiple context-free grammar* (over Δ), for short: $(\Delta\text{-})\text{MCFG}$, is a tuple (N, Δ, S, P) where N is a finite \mathbb{N} -sorted set (*non-terminals*), $S \in N_1$ (*initial non-terminal*), and P is a finite set (*productions*) such that

$$P \subseteq_{\text{fin}} \{(A, f, A_1 \cdots A_\ell) \in N \times F_\Delta \times N^\ell \mid \text{sort}(f) = (\text{sort}(A_1) \cdots \text{sort}(A_\ell), \text{sort}(A)), \\ f \text{ is linear}, \ell \in \mathbb{N}\}.$$

We construe P as an $(N^* \times N)$ -sorted set where for every $\rho = (A, f, A_1 \cdots A_\ell) \in P$ we have $\text{sort}(\rho) = (A, A_1 \cdots A_\ell)$. \square

Let $G = (N, \Delta, S, P)$ be an MCFG and $w \in \Delta^*$. A production $(A, f, A_1 \cdots A_\ell) \in P$ is usually written as $A \rightarrow f(A_1, \dots, A_\ell)$; it inherits rank and fan-out from f . Also, $\text{rank}(G) = \max_{\rho \in P} \text{rank}(\rho)$ and $\text{fan-out}(G) = \max_{\rho \in P} \text{fan-out}(\rho)$. MCFGs of fan-out at most k are called *k-MCFGs*.

The function $\text{yield} : T_P \rightarrow (\Sigma^*)^*$ assigns to every tree $d \in T_P$ the string obtained by projecting every production in d to the contained function (i.e. the second component) and then interpreting the resulting term over F_Δ .

Let $A \in N$. The *set of subderivations in G from A* , denoted by $D_G(A)$, is the set of all terms over P with sort A , i.e. $D_G(A) = (T_P)_A$. The *set of derivations in G* is $D_G = D_G(S)$. Let $w \in \Sigma^*$. The *set of derivations of w in G* is $D_G(w) = \{d \in D_G \mid \text{yield}(d) = (w)\}$.

The *language of G* is $L(G) = \{w \in \Delta^* \mid D_G(w) \neq \emptyset\}$. A language L is called *multiple context-free* if there is an MCFG G with $L = L(G)$. The *set of multiple context-free languages (for which a k-MCFG exists)* is denoted by MCFL (*k-MCFL*, respectively).

The language class *k-MCFL* is a substitution-closed full abstract family of languages [SMFK91, Theorem 3.9]. In particular, *k-MCFL* is closed under intersection with regular languages and under homomorphisms.

Definition 2.5. An \mathcal{A} -*weighted MCFG* (over Δ) is a tuple (N, Δ, S, P, μ) such that (N, Δ, S, P) is an MCFG and $\mu : P \rightarrow \mathcal{A} \setminus \{0\}$ (*weight assignment*). \square

Let $G = (N, \Delta, S, P, \mu)$ be an \mathcal{A} -weighted MCFG and $w \in \Delta^*$. The *set of derivations of w in G* is the set of derivations of w in (N, Δ, S, P) . G inherits fan-out from

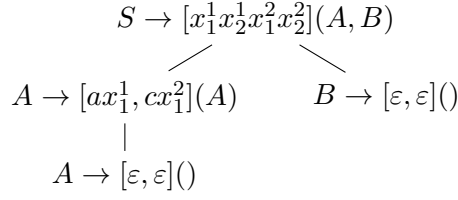


Figure 1: Only derivation of ac in G (Example 2.6)

(N, Δ, S, P) ; \mathcal{A} -weighted MCFGs of fan-out at most k are called \mathcal{A} -weighted k -MCFGs. We define a function $\hat{\mu} : D_G \rightarrow \mathcal{A}$ that applies μ at every position of a given derivation and then multiplies the resulting values (in any order, since \cdot is commutative).

The \mathcal{A} -weighted language induced by G is the function $\llbracket G \rrbracket : \Delta^* \rightarrow \mathcal{A}$ where for every $w \in \Delta^*$ we have $\llbracket G \rrbracket(w) = \sum_{d \in D_G(w)} \hat{\mu}(d)$. Two (\mathcal{A} -weighted) MCFGs are *equivalent* if they induce the same (\mathcal{A} -weighted) language. An \mathcal{A} -weighted language L is called *multiple context-free (of fan-out k)* if there is an \mathcal{A} -weighted k -MCFG G such that $L = \llbracket G \rrbracket$; $k\text{-MCFL}(\mathcal{A})$ denotes the set of multiple context-free \mathcal{A} -weighted languages of fan-out k .

Example 2.6. Consider the Pr_2 -weighted MCFG $G = (N, \Delta, S, P, \mu)$ where $N_1 = \{S\}$, $N_2 = \{A, B\}$, $N = N_1 \cup N_2$, $\Delta = \{a, b, c, d\}$, and P and μ are given by

$$\begin{array}{ll}
P : \quad \rho_1 = S \rightarrow [x_1^1 x_2^1 x_1^2 x_2^2](A, B) & \mu : \quad \mu(\rho_1) = 1 \\
\rho_2 = A \rightarrow [ax_1^1, cx_1^2](A) & \mu(\rho_2) = 1/2 \\
\rho_3 = B \rightarrow [bx_1^1, dx_1^2](B) & \mu(\rho_3) = 1/3 \\
\rho_4 = A \rightarrow [\varepsilon, \varepsilon]() & \mu(\rho_4) = 1/2 \\
\rho_5 = B \rightarrow [\varepsilon, \varepsilon]() & \mu(\rho_5) = 2/3 .
\end{array}$$

We observe that $\text{supp}(\llbracket G \rrbracket) = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$ and for every $m, n \in \mathbb{N}$ we have

$$\begin{aligned}
\llbracket G \rrbracket(a^m b^n c^m d^n) &= \mu(\rho_1) \cdot (\mu(\rho_2))^m \cdot \mu(\rho_4) \cdot (\mu(\rho_3))^m \cdot \mu(\rho_5) \\
&= 1/(2^m \cdot 3^{n+1}).
\end{aligned}$$

The only derivation of $w = ac$ in G is shown in Figure 1, its weight and hence also the weight of w is $1/(2^1 \cdot 3^{0+1}) = 1/6$. \square

A non-terminal is called *productive in an (\mathcal{A} -weighted) MCFG* if there is at least one subderivation starting from this non-terminal. It is obvious that every (\mathcal{A} -weighted) k -MCFL can be recognised by an (\mathcal{A} -weighted) k -MCFG that only has productive non-terminals.

Non-deleting normal form

An (\mathcal{A} -weighted) MCFG is called *non-deleting* if the string function in every production is linear and non-deleting. Seki, Matsumura, Fujii, and Kasami [SMFK91, Lemma 2.2]

proved that for every k -MCFG there is an equivalent non-deleting k -MCFG. We generalise this to \mathcal{A} -weighted MCFGs.

Lemma 2.7. For every \mathcal{A} -weighted k -MCFG there is an equivalent non-deleting \mathcal{A} -weighted k -MCFG.

Proof idea. We modify the construction for the unweighted case [SMFK91, Lemma 2.2] such that it preserves the structure of derivations. Then a weight assignment can be defined in an obvious manner.

Proof. Let $G = (N, \Delta, S, P, \mu)$ be an \mathcal{A} -weighted k -MCFG. When examining the proof of Seki, Matsumura, Fujii, and Kasami [SMFK91, Lemma 2.2], we notice that only step 2 of Procedure 1 deals with non-deletion. We construct N' and P' from (N, Δ, S, P) by step 2 of Procedure 1, but drop the restriction that $\Psi \neq [\text{sort}(A)]$.¹

Let $g : P' \rightarrow P$ assign to every $\rho' \in P'$ the production in G it has been constructed from. Furthermore, let $\widehat{g} : D_{G'} \rightarrow D_G$ be the function obtained by applying g point-wise. We show the following hypothesis by induction on the structure of subderivations:

Induction hypothesis: For every $A \in N$ and $\Psi \in M(A) : \widehat{g}$ is a bijection between $D_{G'}(A[\Psi])$ and $D_G(A)$.

Induction step: Let $d \in D_G(A)$ and $\Psi \in M(A)$ with $d = \rho(d_1, \dots, d_k)$ for some production $\rho \in P$ and derivations $d_1 \in D_G(A_1), \dots, d_k \in D_G(A_k)$. The construction defines $\Psi_1 \in M(A_1), \dots, \Psi_k \in M(A_k)$ and a production ρ' which is unique for every ρ and Ψ . By the induction hypothesis, we know that there are derivations d'_1, \dots, d'_k which are unique for $(d_1, \Psi_1), \dots, (d_k, \Psi_k)$, respectively. Therefore, $d' = \rho(d'_1, \dots, d'_k)$ is unique for d and Ψ . Hence for every Ψ , \widehat{g} induces a bijection between $D_{G'}(A[\Psi])$ and $D_G(A)$.

By construction, the new start symbol is $S[\emptyset]$; hence for the elements of $D_{G'}$, we set $\Psi = \emptyset$ and by induction hypothesis we obtain that \widehat{g} is bijective. Since \widehat{g} preserves the structure of derivations and is a bijection we obtain $\widehat{\mu \circ g} = \widehat{\mu} \circ \widehat{g}$. Hence $\llbracket (N', \Delta, S[\emptyset], P', \mu \circ g) \rrbracket = \llbracket G \rrbracket$. The fan-out is not increased by this construction. ■

3 Multiple Dyck languages

According to Kanazawa [Kan14, Section 1] there is no definition of multiple Dyck languages using congruence relations. We close this gap by giving such a definition (Definition 3.3).

3.1 The original definition

We recall the definition of multiple Dyck languages [YKS10, Definition 1]:

Definition 3.1. Let Δ be a finite \mathbb{N} -sorted set,² $\overline{(\cdot)}$ be a bijection between Δ and some alphabet $\overline{\Delta}$, $k = \max_{\delta \in \Delta} \text{sort}(\delta)$, and $r \geq k$. The *multiple Dyck grammar with respect*

¹This construction may therefore create productions of fan-out 0.

²In Yoshinaka, Kaji, and Seki [YKS10], \mathbb{N} -sorted sets are called indexed sets and sort is denoted as dim .

to Δ is the k -MCFG $G_\Delta = (\{A_1, \dots, A_k\}, \hat{\Delta}, A_1, P)$ where $\hat{\Delta} = \{\delta^{[i]}, \bar{\delta}^{[i]} \mid \delta \in \Delta, i \in [\text{sort}(\delta)]\}$, $\text{sort}(A_i) = i$ for every $i \in [k]$, and P is the smallest set such that

- (i) for every linear non-deleting³ terminal-free string function $f \in (F_\Delta)_{(s_1 \dots s_\ell, s)}$ with $\ell \in [r]$ and $s_1, \dots, s_\ell, s \in [k]$ we have

$$A_s \rightarrow f(A_{s_1}, \dots, A_{s_\ell}) \in P,$$

- (ii) for every $\delta \in \Delta$ with $\text{sort } s$ we have

$$A_s \rightarrow [\delta^{[1]}x_1^1\bar{\delta}^{[1]}, \dots, \delta^{[s]}x_1^s\bar{\delta}^{[s]}](A_s) \in P, \text{ and}$$

- (iii) for every $s \in [k]$ we have

$$A_s \rightarrow [u_1, \dots, u_s](A_s) \in P$$

where $u_i \in \{x_i, x_i\delta^{[1]}\bar{\delta}^{[1]}, \delta^{[1]}\bar{\delta}^{[1]}x_i \mid \delta \in \Delta_1\}$ for every $i \in [s]$.

The *multiple Dyck language with respect to Δ* , denoted by $mD(\Delta)$, is $L(G_\Delta)$. We call $\max_{\delta \in \Delta} \text{sort}(\delta)$ the *dimension of $mD(\Delta)$* . The *set of multiple Dyck languages of dimension at most k* is denoted by $k\text{-mDYCK}$. \square

3.2 Congruence multiple Dyck languages

For the rest of this section let Σ be an alphabet. Also let $\overline{\Sigma}$ be a set (disjoint from Σ) and $\overline{(\cdot)}$ be a bijection between Σ and $\overline{\Sigma}$. Intuitively Σ and $\overline{\Sigma}$ are sets of opening and closing parentheses and $\overline{(\cdot)}$ matches an opening to its closing parenthesis.

We define \equiv_Σ as the smallest congruence relation on the free monoid $(\Sigma \cup \overline{\Sigma})^*$ where for every $\sigma \in \Sigma$ the cancellation rule $\sigma\overline{\sigma} \equiv_\Sigma \varepsilon$ holds. The *Dyck language with respect to Σ* , denoted by $D(\Sigma)$, is $[\varepsilon]_{\equiv_\Sigma}$. The *set of Dyck languages* is denoted by DYCK .

Example 3.2. Let $\Sigma = \{(\langle, [, \llbracket\}$. We abbreviate $\overline{(\langle, \overline{[\langle, \overline{[\llbracket}$ by $\rangle, \rangle, \rrbracket$ and \rrbracket by $\rangle, \rangle, \rrbracket$, respectively. Then we have for example $\llbracket(\rangle)\langle() \equiv_\Sigma \llbracket\langle\rangle \equiv_\Sigma \llbracket \equiv_\Sigma \varepsilon$ and $(\llbracket\rrbracket)\langle() \equiv_\Sigma (\llbracket\rrbracket) \equiv_\Sigma (\llbracket\rrbracket) \not\equiv_\Sigma \varepsilon$. \square

Let \mathfrak{P} be a partition of Σ . We define $\equiv_{\Sigma, \mathfrak{P}}$ as the smallest congruence relation on the free monoid $(\Sigma \cup \overline{\Sigma})^*$ such that if $v_1 \dots v_\ell \equiv_{\Sigma, \mathfrak{P}} \varepsilon$ with $v_1, \dots, v_\ell \in D(\Sigma)$, then the *cancellation rule*

$$u_0\sigma_1v_1\overline{\sigma_1}u_1 \dots \sigma_\ell v_\ell \overline{\sigma_\ell} u_\ell \equiv_{\Sigma, \mathfrak{P}} u_0 \dots u_\ell$$

holds for every $\{\sigma_1, \dots, \sigma_\ell\} \in \mathfrak{P}$ and $u_0, \dots, u_\ell \in D(\Sigma)$. Intuitively, every element of \mathfrak{P} denotes a set of *linked* opening parentheses, i.e. parentheses that must be consumed simultaneously by $\equiv_{\Sigma, \mathfrak{P}}$.

³We add the restriction “non-deleting” in comparison to the original definition since the proof of Lemma 1 in Yoshinaka, Kaji, and Seki [YKS10] only uses non-deleting rules.

Definition 3.3. The *congruence multiple Dyck language with respect to Σ and \mathfrak{P}* , denoted by $mD_c(\Sigma, \mathfrak{P})$, is $[\varepsilon]_{\equiv_{\Sigma, \mathfrak{P}}}$. \square

Example 3.4. Let $\Sigma = \{(\langle, \lfloor, \llbracket\}$ and $\mathfrak{P} = \{\mathfrak{p}_1, \mathfrak{p}_2\}$ where $\mathfrak{p}_1 = \{(\langle, \rangle\}$ and $\mathfrak{p}_2 = \{[\lfloor, \rrbracket\}$. We abbreviate $\bar{(\langle, \rangle}$, $\bar{[\lfloor, \rrbracket}$, and \llbracket by \rangle , \rrbracket , \llbracket , respectively. Then we have for example $\llbracket() \rrbracket \langle \rangle \equiv_{\Sigma, \mathfrak{P}} \varepsilon$ since $\mathfrak{p}_2 = \{[\lfloor, \rrbracket\} \in \mathfrak{P}$, $() \langle \rangle \equiv_{\Sigma, \mathfrak{P}} \varepsilon$, and $u_0 = u_1 = u_2 = \varepsilon$. But $\llbracket() \rrbracket \langle \rangle \not\equiv_{\Sigma, \mathfrak{P}} \varepsilon$ since when instantiating the cancellation rule with any of the two elements of \mathfrak{P} , we can not reduce $\llbracket() \rrbracket \langle \rangle$:

- (i) If we choose $\{\sigma_1, \sigma_2\} = \{[\lfloor, \rrbracket\}$ then we would need to set $u_1 = \langle$ and $u_2 = \rangle$, but they are not in $D(\Sigma)$, also $() \not\equiv_{\Sigma, \mathfrak{P}} \varepsilon$;
- (ii) If we choose $\{\sigma_1, \sigma_2\} = \{(\langle, \rangle\}$ then we would need to set $u_0 = \llbracket$ and $u_1 = \rrbracket$, but they are not in $D(\Sigma)$, also $\llbracket \not\equiv_{\Sigma, \mathfrak{P}} \varepsilon$.

Hence $\llbracket() \rrbracket \langle \rangle, () \langle \rangle \in mD_c(\Sigma, \mathfrak{P})$ and $\llbracket() \rrbracket \langle \rangle \notin mD_c(\Sigma, \mathfrak{P})$. \square

Observation 3.5. From the definition of $\equiv_{\Sigma, \mathfrak{P}}$ it is easy to see that for every $u_1, \dots, u_k \in D(\Sigma)$ and $v_1, \dots, v_\ell \in D(\Sigma)$ we have that $u_1 \dots u_k, v_1 \dots v_\ell \in mD_c(\Sigma, \mathfrak{P})$ implies that every permutation of $u_1, \dots, u_k, v_1, \dots, v_\ell$ is in $mD_c(\Sigma, \mathfrak{P})$. \blacksquare

The *dimension* of $mD_c(\Sigma, \mathfrak{P})$ is $\max_{\mathfrak{p} \in \mathfrak{P}} |\mathfrak{p}|$. The *set of congruence multiple Dyck languages (of at most dimension k)* is denoted by mDYCK_c ($k\text{-mDYCK}_c$, respectively).

Proposition 3.6. $k\text{-mDYCK} \subseteq k\text{-mDYCK}_c$

Proof. We show that a tuple (w_1, \dots, w_m) can be generated in G_Δ from non-terminal A_m if and only if w_1, \dots, w_m are all Dyck words and $w_1 \dots w_m$ is a multiple Dyck word. The “only if” we prove by induction on the structure of derivations in G_Δ . For “if” we construct derivations in G_Δ by induction on the number of applications of the cancellation rule (including the number of applications to reduce the word $v_1 \dots v_\ell$ from the definition on the cancellation rule to ε).

Let $mD \in k\text{-mDYCK}$. Then there is an \mathbb{N} -sorted set Δ such that $mD = mD(\Delta)$ and $k \geq \max_{\delta \in \Delta} \text{sort}(\delta)$. We define $\mathfrak{p}_\delta = \{\delta^{[i]} \mid i \in [\text{sort}(\delta)]\}$ for every $\delta \in \Delta$, $\Sigma = \bigcup_{\delta \in \Delta} \mathfrak{p}_\delta$, and $\mathfrak{P} = \{\mathfrak{p}_\delta \mid \delta \in \Delta\}$. Clearly $\max_{\mathfrak{p} \in \mathfrak{P}} |\mathfrak{p}| \leq k$. Thus $mD_c(\Sigma, \mathfrak{P}) \in k\text{-mDYCK}$. Let $\text{Tup}(G_\Delta, A)$ denote the set of tuples generated in G_Δ when starting with non-terminal A where A is not necessarily initial. In the following we show that for every $m \in [\max_{\delta \in \Delta} \text{sort}(\delta)]$ and $w_1, \dots, w_m \in (\Sigma \cup \bar{\Sigma})^*$:

$$(w_1, \dots, w_m) \in \text{Tup}(G_\Delta, A_m) \iff w_1 \dots w_m \in mD_c(\Sigma, \mathfrak{P}) \wedge w_1, \dots, w_m \in D(\Sigma) \quad (*)$$

(\Rightarrow) It follows from the definitions of Tup and G_Δ that $(w_1, \dots, w_m) \in \text{Tup}(G_\Delta, A_m)$ implies that there are a rule $A_m \rightarrow f(A_{m_1}, \dots, A_{m_\ell})$ in G_Δ and a tuple $\vec{u}_i = (u_i^1, \dots, u_i^{m_i})$ in $\text{Tup}(G_\Delta, A_{m_i})$ for every $i \in [\ell]$ such that $f(\vec{u}_1, \dots, \vec{u}_\ell) = (w_1, \dots, w_m)$. By applying the induction hypothesis ℓ times, we also have that $u_1^1, \dots, u_1^{m_1}, \dots, u_\ell^1, \dots, u_\ell^{m_\ell} \in D(\Sigma)$ and $u_1^1 \dots u_1^{m_1}, \dots, u_\ell^1 \dots u_\ell^{m_\ell} \in mD(\Sigma, \mathfrak{P})$. We distinguish three cases (each corresponding to one type of rule in G_Δ):

- (i) f is linear, non-deleting, and terminal-free. Then we have for every $i \in [m]$ that $w_i \in \{u_1^1, \dots, u_1^{m_1}, \dots, u_\ell^1, \dots, u_\ell^{m_\ell}\}^*$ and therefore also $w_i \in D(\Sigma)$. Furthermore, by applying Observation 3.5 $(\ell - 1)$ times, we have that $w_1 \cdots w_m \in mD_c(\Sigma, \mathfrak{P})$.
- (ii) $f = [\delta^{[1]}x_1^1\bar{\delta}^{[1]}, \dots, \delta^{[m]}x_1^m\bar{\delta}^{[m]}]$; then $\ell = 1$, $m_1 = m$, and for every $i \in [m]$ we have $w_i = \delta^{[i]}u_1^i\bar{\delta}^{[i]}$ and since $u_1^i \in D(\Sigma)$ also $w_i \in D(\Sigma)$. Furthermore, $w_1 \cdots w_m = \delta^{[1]}u_1^1\bar{\delta}^{[1]} \dots \delta^{[m]}u_1^m\bar{\delta}^{[m]} \in mD_c(\Sigma, \mathfrak{P})$ due to the cancellation rule.
- (iii) $f = [u_1, \dots, u_m]$ where $u_i \in \{x_i^1, x_i^1\delta^{[1]}\bar{\delta}^{[1]}, \delta^{[1]}\bar{\delta}^{[1]}x_i^1 \mid \delta \in \Delta_1\}$ for every $i \in [m]$; then $w_i \in \{u_i^1, u_i^1\delta^{[1]}\bar{\delta}^{[1]}, \delta^{[1]}\bar{\delta}^{[1]}u_i^1 \mid \delta \in \Delta_1\}$ for every $i \in [m]$, $\ell = 1$, and $m_1 = m$. Since \equiv_Σ is a congruence relation (in particular, \equiv_Σ respects composition), we have that $w_1, \dots, w_m \in D(\Sigma)$. By applying Observation 3.5 m times, we have that $w_1 \cdots w_m \in mD_c(\Sigma, \mathfrak{P})$.

(\Leftarrow) If the cancellation rule is applied zero times in order to reduce $w_1 \cdots w_m$ to ε then $w_1 = \dots = w_m = \varepsilon$. The rule $A_m \rightarrow [\varepsilon, \dots, \varepsilon]()$ in G_D clearly derives (w_1, \dots, w_m) . If the cancellation rule is applied $i + 1$ times in order to reduce $w_1 \cdots w_m$ to ε then $w_1 \cdots w_m$ has the form $u_0\sigma_1v_1\bar{\sigma}_1u_1 \cdots \sigma_\ell v_\ell \bar{\sigma}_\ell u_\ell$ for some $u_0, \dots, u_\ell \in D(\Sigma)$, $v_1, \dots, v_\ell \in D(\Sigma)$, and $\{\sigma_1, \dots, \sigma_\ell\} \in \mathfrak{P}$ with $v_1 \cdots v_\ell \equiv_{\Sigma, \mathfrak{P}} \varepsilon$. Then we need to apply the cancellation rule at most i times to reduce $v_1 \cdots v_\ell$ to ε , hence, by induction hypothesis, there is some $d \in D_{G_\Delta}$ that derives (v_1, \dots, v_ℓ) . We use an appropriate rule ρ of type (ii) such that $\rho(d)$ derives $(\sigma_1v_1\bar{\sigma}_1, \dots, \sigma_\ell v_\ell \bar{\sigma}_\ell)$. Also, we need to apply the cancellation rule at most i times in order to reduce $u_0 \cdots u_\ell$ to ε , hence, by induction hypothesis, there are derivations d_1, \dots, d_n that derive tuples containing exactly u_0, \dots, u_ℓ as components. Then there is a rule ρ' of type (iii) such that $\rho'(\rho(d), d_1, \dots, d_n) \in D_{G_\Delta}$ derives the tuple (w_1, \dots, w_m) .

From (*) with $m = 1$ and the fact “ $w_1 \in mD_c(\Sigma, \mathfrak{P})$ implies $w_1 \in D(\Sigma)$ ” we get that $mD_c(\Sigma, \mathfrak{P}) = mD$. ■

Lemma 3.7. $k\text{-mDYCK}_c \subseteq k\text{-MCFL}$

Proof idea. For a given congruence multiple Dyck language L , we construct a multiple Dyck grammar that is equivalent up to a homomorphism g . We then use the closure of $k\text{-MCFL}$ under homomorphisms.

Proof. Let $L \in k\text{-mDYCK}_c$. Then there are an alphabet Σ and a partition \mathfrak{P} of Σ such that $mD_c(\Sigma, \mathfrak{P}) = L$. Consider \mathfrak{P} as an \mathbb{N} -sorted set where the sort of an element is its cardinality. Then $\widehat{\Delta} = \{\mathbf{p}^{[i]}, \bar{\mathbf{p}}^{[i]} \mid \mathbf{p} \in \mathfrak{P}, i \in [|\mathbf{p}|]\}$. For every $\mathbf{p} \in \mathfrak{P}$ assume some fixed enumeration of the elements of \mathbf{p} . We define a bijection $g : \widehat{\Delta} \rightarrow \Sigma \cup \overline{\Sigma}$ such that every $\mathbf{p}^{[i]}$ (for some \mathbf{p} and i) is assigned the i -th element of \mathbf{p} and $g(\bar{\mathbf{p}}^{[i]}) = g(\mathbf{p}^{[i]})$.

Let $\text{Tup}_g(G_{\mathfrak{P}})$ denote the set of tuples obtained by interpreting the terms corresponding to every subderivation in $G_{\mathfrak{P}}$ and then applying g to every component. We show the following claim by induction:

$$\begin{aligned}
w \in mD(\mathfrak{P}) &\iff \forall \ell \in \mathbb{N}, u_0, \dots, u_\ell, w_1, \dots, w_\ell \in D(\Sigma) \\
&\text{with } w = u_0 w_1 u_1 \cdots w_\ell u_\ell : \\
&(u_0, w_1, u_1, \dots, w_\ell, u_\ell) \in \text{Tup}_g(G_{\mathfrak{P}}) \tag{IH}
\end{aligned}$$

Note that in the following the indices of the elements of $\mathbf{p} = \{\sigma_1, \dots, \sigma_\ell\}$ are chosen such that they respect the previously fixed enumeration of \mathbf{p} , i.e. for every $i \in [\ell] : g(\mathbf{p}^{[i]}) = \sigma_i$. We derive

$$\begin{aligned}
& w \in mD(\mathfrak{P}) \\
& \iff \forall \ell \in \mathbb{N}, u_0, \dots, u_\ell, v_1, \dots, v_\ell \in D(\Sigma), \mathbf{p} = \{\sigma_1, \dots, \sigma_\ell\} \in \mathfrak{P} \text{ with} \\
& \quad w = u_0 \sigma_1 v_1 \overline{\sigma_1} u_1 \cdots \sigma_\ell v_\ell \overline{\sigma_\ell} u_\ell : u_0 v_1 u_1 \cdots v_\ell u_\ell \in mD(\mathfrak{P}) \quad (\text{by def. of } mD(\mathfrak{P})) \\
& \iff \forall \ell \in \mathbb{N}, u_0, \dots, u_\ell, v_1, \dots, v_\ell \in D(\Sigma), \mathbf{p} = \{\sigma_1, \dots, \sigma_\ell\} \in \mathfrak{P} \text{ with} \\
& \quad w = u_0 \sigma_1 v_1 \overline{\sigma_1} u_1 \cdots \sigma_\ell v_\ell \overline{\sigma_\ell} u_\ell : (u_0, v_1, u_1, \dots, v_\ell, u_\ell) \in \text{Tuple}_g(G_{\mathfrak{P}}) \quad (\text{by (IH)}) \\
& \iff \forall \ell \in \mathbb{N}, u_0, \dots, u_\ell, v_1, \dots, v_\ell \in D(\Sigma), \mathbf{p} = \{\sigma_1, \dots, \sigma_\ell\} \in \mathfrak{P} \text{ with} \\
& \quad w = u_0 \sigma_1 v_1 \overline{\sigma_1} u_1 \cdots \sigma_\ell v_\ell \overline{\sigma_\ell} u_\ell : \\
& \quad (u_0, \sigma_1 v_1 \overline{\sigma_1}, u_1, \dots, \sigma_\ell v_\ell \overline{\sigma_\ell}, u_\ell) \in \text{Tuple}_g(G_{\mathfrak{P}}) \quad (\text{by def. of } G_{\mathfrak{P}}) \\
& \iff \forall \ell \in \mathbb{N}, u_0, \dots, u_\ell, w_1, \dots, w_\ell \in D(\Sigma) \text{ with } w = u_0 w_1 u_1 \cdots w_\ell u_\ell : \\
& \quad (u_0, w_1, u_0, \dots, w_\ell, u_\ell) \in \text{Tuple}_g(G_{\mathfrak{P}}) \quad (\text{using permuting productions in } G_{\mathfrak{P}})
\end{aligned}$$

$g(L(G_{\mathfrak{P}})) = L$ follows by instantiating (IH) for $\ell = 0$ and discovering that $\{t \mid (t) \in \text{Tuple}_g(G_{\mathfrak{P}})\} = g(L(G_{\mathfrak{P}}))$.

Since k -MCFLs are closed under homomorphisms [SMFK91, Theorem 3.9], we know that $L \in k\text{-MCFL}$.⁴ ■

Observation 3.8. Examining the definition of multiple Dyck grammars, we observe that some production in Item (ii) has fan-out k for at least one $\delta \in \Delta$. Then, using Seki, Matsumura, Fujii, and Kasami [SMFK91, Theorem 3.4], we have for every $k \geq 1$ that $(k+1)\text{-mDYCK}_c \setminus k\text{-MCFL} \neq \emptyset$. ■

Proposition 3.9. $\text{DYCK} = 1\text{-mDYCK}_c \subsetneq 2\text{-mDYCK}_c \subsetneq \dots$

Proof. We get ' \subseteq ' from the definition of $k\text{-mDYCK}_c$ and ' \neq ' from Observation 3.8. We have the equality since the dimension of some partition \mathfrak{P} of Σ is 1 if and only if $\mathfrak{P} = \{\{\sigma\} \mid \sigma \in \Sigma\}$. Then we have $\equiv_\Sigma = \equiv_{\Sigma, \mathfrak{P}}$ and thus $D(\Sigma) = mD_c(\Sigma, \mathfrak{P})$. Hence $\text{DYCK} = 1\text{-mDYCK}_c$. ■

3.3 Membership in a congruence multiple Dyck language

We provide a recursive algorithm (Algorithm 3) to decide whether a word w is in a given congruence multiple Dyck language $mD_c(\Sigma, \mathfrak{P})$. This amounts to checking whether $w \equiv_{\Sigma, \mathfrak{P}} \varepsilon$, and it suffices to only apply the cancellation rule from left to right.

In order for Algorithm 3 to decide the membership in a multiple Dyck language it must consider all decompositions of the input string into Dyck words. For this purpose we define a function *split* that decomposes a given Dyck word into *shortest, non-empty* Dyck words.

⁴This construction shows that congruence multiple Dyck languages (Definition 3.3) are equivalent to multiple Dyck languages [YKS10, Definition 1] up to the application of g .

The function *split*

As Dyck languages are recognisable by pushdown automata, we define a data structure pushdown and two functions with side-effects on pushdowns. A *pushdown* is a string over some alphabet Γ . Let Γ be an alphabet, $\gamma \in \Gamma$, and $pd \subseteq \Gamma^*$ be a pushdown.

- $\text{pop}(pd)$ returns the left-most symbol of pd and removes it from pd .
- $\text{push}(pd, \gamma)$ prepends γ to pd .

Note that $\text{pop}()$ is only a partial function, it is undefined for $pd = \varepsilon$. But since the input word w is required to be in $D(\Sigma)$ by Algorithm 2, the expression on line 6 is always defined.

Algorithm 2 Algorithm to split a word in $D(\Sigma)$ into shortest non-empty strings from $D(\Sigma)$

Input: alphabet Σ , Dyck word $w \in D(\Sigma)$

Output: sequence (u_1, \dots, u_ℓ) of shortest, non-empty Dyck words with $w = u_1 \cdots u_\ell$

```

1: function split( $\Sigma, w$ )
2:   let  $pd = \varepsilon$ ,  $j = 1$ , and  $u_j = \varepsilon$ 
3:   for  $0 \leq i \leq |w|$  do
4:     append  $w_i$  to  $u_j$ 
5:     if  $w_i \in \overline{\Sigma}$  then
6:        $\text{pop}(pd)$ 
7:       if  $pd = \varepsilon$  then
8:         increase  $j$  by 1 and let  $u_j = \varepsilon$ 
9:       end if
10:    else
11:       $\text{push}(pd, \overline{w_i})$ 
12:    end if
13:  end for
14:  return  $(u_1, \dots, u_{j-1})$ 
15: end function

```

One can easily see that *split* is bijective (the inverse function is concatenation). We therefore say that w and (u_1, \dots, u_ℓ) correspond to each other, and for every operation on either of them there is a corresponding operation on the other. In particular the empty string corresponds to the empty tuple.

Outline of the function *isMember*

If w is the empty word, we return 1 on line 2 since the empty word is in $mD_c(\Sigma, \mathfrak{P})$. Then we check if w is in $D(\Sigma)$, e.g. with the context-free grammar in (7.6) in Salomaa [Sal73]. If w is not in $D(\Sigma)$, it is also not in $mD_c(\Sigma, \mathfrak{P})$ and we return 0. Otherwise, we split w into shortest non-empty Dyck words (on line 4) using the function *split*. Since each of

Algorithm 3 Function *isMember* to decide membership in $mD_c(\Sigma, \mathfrak{P})$

Input: Σ , \mathfrak{P} , and $w \in (\Sigma \cup \overline{\Sigma})^*$

Output: 1 if $w \in mD_c(\Sigma, \mathfrak{P})$, 0 otherwise

```

1: function isMember( $\Sigma, \mathfrak{P}, w$ )
2:   if  $w = \varepsilon$  then return 1 end if
3:   if  $w \notin D(\Sigma)$  then return 0 end if
4:    $(\sigma_1 u_1 \overline{\sigma}_1, \dots, \sigma_\ell u_\ell \overline{\sigma}_\ell) \leftarrow \text{split}(\Sigma, w)$   $\triangleright$  such that  $\sigma_1, \dots, \sigma_\ell \in \Sigma$ 
5:    $\mathcal{I} \leftarrow \{I \subseteq \mathcal{P}([ \ell ]) \mid I \text{ partition of } [ \ell ], \forall \{i_1, \dots, i_k\} \in I: \{\sigma_{i_1}, \dots, \sigma_{i_k}\} \in \mathfrak{P}\}$ 
6:   for  $I \in \mathcal{I}$  do
7:      $b \leftarrow 1$ 
8:     for  $\{i_1, \dots, i_k\} \in I$  do  $\triangleright$  such that  $i_1 < \dots < i_k$ 
9:        $b \leftarrow b \cdot \text{isMember}(\Sigma, \mathfrak{P}, u_{i_1} \dots u_{i_k})$ 
10:    end for
11:    if  $b = 1$  then return 1 end if
12:  end for
13:  return 0
14: end function

```

those shortest non-empty Dyck words has the form $\sigma u \overline{\sigma}$ for some $\sigma \in \Sigma$ and $u \in (\Sigma \cup \overline{\Sigma})^*$, we write $(\sigma_1 u_1 \overline{\sigma}_1, \dots, \sigma_\ell u_\ell \overline{\sigma}_\ell)$ for the left-hand side of the assignment on line 4. On line 5 we calculate the set \mathcal{I} of all partitions I such that each element of I specifies a set of components of the tuple $(\sigma_1 u_1 \overline{\sigma}_1, \dots, \sigma_\ell u_\ell \overline{\sigma}_\ell)$ whose outer parentheses can be removed with one application of the cancellation rule. Since I is a partition, we know that the outer parentheses of every component can be removed via the cancellation rule. Then it remains to be shown that there is a partition I such that for each element $\{i_1, \dots, i_k\}$ of I the word $u_{i_1} \dots u_{i_k}$ is an element of $mD_c(\Sigma, \mathfrak{P})$; this is done on lines 6 to 12. If there is no such partition, then we return 0 on line 13.

Example 3.10 (Example 3.4 continued). Table 4 shows a run of Algorithm 3 on the word $\llbracket () \rrbracket \llbracket \langle \rangle \rrbracket$ where we report return values and a subset of the variable assignment whenever we reach the end of lines 4, 5, 6, 8, 9. The recursive calls to *isMember* are indented. Table 5 shows the run of Algorithm 3 on the word $\llbracket () \rrbracket \llbracket \langle \rangle \rrbracket \llbracket \langle \rangle \rrbracket$. \square

In light of the close link between Algorithm 3 and the relation $\equiv_{\Sigma, \mathfrak{P}}$ we omit the proof of correctness.

Proof of termination for Algorithm 3. If $w = \varepsilon$, the algorithm terminates on line 2. If $w \notin D(\Sigma)$, the algorithm terminates on line 3. Since \mathcal{I} is finite and each element $I \in \mathcal{I}$ is also finite, there are only finitely many calls to *isMember* on line 9 for each recursion. In each of those calls, the length of the third argument is strictly smaller than the length of w . Therefore, after a finite number of recursions, the third argument passed to *isMember* is either the empty word, then the algorithm terminates on line 2, or not an element of $D(\Sigma)$, then the algorithm terminates on line 2. \blacksquare

Table 4: Run of Algorithm 3 on the word $\llbracket () \rrbracket \langle \rangle$, cf. Examples 3.4 and 3.10.

```

isMember( $\Sigma, \mathfrak{P}, \llbracket () \rrbracket \langle \rangle$ )
l. 4:  $\sigma_1 = \llbracket, \sigma_2 = [, u_1 = (), u_2 = \langle$ 
l. 5:  $\mathcal{I} = \{\{\{1, 2\}\}\}$ 
l. 6:  $I = \{\{1, 2\}\}$ 
l. 8:  $k = 2, i_1 = 1, i_2 = 2$ 
l. 9:  $b = 1 \cdot \text{isMember}(\Sigma, \mathfrak{P}, () \langle \rangle)$ 
      l. 4:  $\sigma_1 = (, \sigma_2 = \langle, u_1 = \varepsilon = u_2$ 
      l. 5:  $\mathcal{I} = \{\{\{1, 2\}\}\}$ 
      l. 6:  $I = \{\{1, 2\}\}$ 
      l. 8:  $k = 2, i_1 = 1, i_2 = 2$ 
      l. 9:  $b = 1 \cdot \text{isMember}(\Sigma, \mathfrak{P}, \varepsilon)$ 
            l. 2: return 1
      l. 9:  $b = 1 \cdot 1 = 1$ 
      l. 11: return 1
l. 9:  $b = 1 \cdot 1$ 
l. 11: return 1

```

4 CS theorem for weighted MCFLs

In this section we generalise the CS representation of (unweighted) MCFLs [YKS10, Theorem 3] to the weighted case. We prove that an \mathcal{A} -weighted MCFL L can be decomposed into an \mathcal{A} -weighted alphabetic homomorphism h , a regular language R and a congruence multiple Dyck language mD_c such that $L = h(R \cap mD_c)$.

To show this, we use the proof idea from Droste and Vogler [DV13]: We separate the weight from our grammar formalism and then use the unweighted CS representation on the unweighted part. The outline of our proof is as follows:

- (i) We separate the weights from L (Lemma 4.3), obtaining an MCFL L' and a weighted alphabetic homomorphism.
- (ii) We use a corollary of the CS representation of (unweighted) MCFLs (Corollary 4.6) to obtain a CS representation of L' .
- (iii) Using the two previous points and a lemma for the composition of weighted and unweighted alphabetic homomorphisms (Lemma 4.8), we obtain a CS representation of L (Theorem 4.10).

Figure 6 outlines the proof of Theorem 4.10. The boxes represent sub-diagrams for which the corresponding lemma proofs existence of the arrows and commutativity.

Table 5: Run of Algorithm 3 on the word $\llbracket () \rrbracket \llbracket \rrbracket \llbracket \langle \rangle \rrbracket$.

```

isMember( $\Sigma, \mathfrak{P}, \llbracket () \rrbracket \llbracket \rrbracket \llbracket \langle \rangle \rrbracket$ )
1.4:  $\sigma_1 = \llbracket = \sigma_3, \sigma_2 = \llbracket = \sigma_4, u_1 = () , u_2 = \varepsilon = u_3, u_4 = \langle \rangle$ 
1.5:  $\mathcal{I} = \{ \{ \{1, 2\}, \{3, 4\} \}, \{ \{1, 4\}, \{2, 3\} \} \}$ 

1.6:  $I = \{ \{1, 2\}, \{3, 4\} \}$ 
1.8:  $k = 2, i_1 = 1, i_2 = 2$ 
1.9:  $b = 1 \cdot \text{isMember}(\Sigma, \mathfrak{P}, ())$ 
      1.4:  $\sigma_1 = (, u_1 = \varepsilon$ 
      1.5:  $\mathcal{I} = \emptyset$ 
      1.13: return 0
1.9:  $b = 1 \cdot 0 = 0$ 
1.8:  $k = 2, i_1 = 3, i_2 = 4$ 
1.9:  $b = 0 \cdot \text{isMember}(\Sigma, \mathfrak{P}, \langle \rangle)$ 
      1.4:  $\sigma_1 = \langle, u_1 = \varepsilon$ 
      1.5:  $\mathcal{I} = \emptyset$ 
      1.13: return 0
1.9:  $b = 0 \cdot 0 = 0$ 

1.6:  $I = \{ \{1, 4\}, \{2, 3\} \}$ 
1.8:  $k = 2, i_1 = 1, i_2 = 4$ 
1.9:  $b = 1 \cdot \text{isMember}(\Sigma, \mathfrak{P}, () \langle \rangle)$ 
      1.4:  $\sigma_1 = (, \sigma_2 = \langle, u_1 = \varepsilon = u_2$ 
      1.5:  $\mathcal{I} = \{ \{ \{1, 2\} \} \}$ 
      1.6:  $I = \{ \{1, 2\} \}$ 
      1.8:  $b = 1 \cdot \text{isMember}(\Sigma, \mathfrak{P}, \varepsilon)$ 
            1.2: return 1
      1.8:  $b = 1 \cdot 1$ 
      1.11: return 1
1.9:  $b = 1 \cdot 1$ 
1.8:  $k = 2, i_1 = 2, i_2 = 3$ 
1.9:  $b = 1 \cdot \text{isMember}(\Sigma, \mathfrak{P}, \varepsilon)$ 
      1.2: return 1
1.9:  $b = 1 \cdot 1$ 

1.11: return 1

```


Lemma 4.8

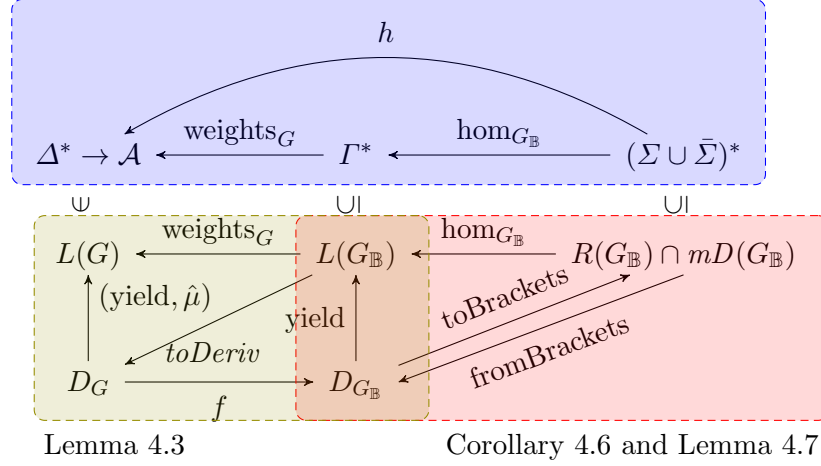


Figure 6: Outline of the proof of Theorem 4.10

4.1 Separating the weights

We split a given weighted MCFG G into an unweighted MCFG $G_{\mathbb{B}}$ and a weighted homomorphism weights_G such that $\llbracket G \rrbracket = \text{weights}_G(L(G_{\mathbb{B}}))$.

Definition 4.1. Let $G = (N, \Delta, S, P, \mu)$ be a non-deleting \mathcal{A} -weighted k -MCFG. The *unweighted MCFG for G* is the non-deleting k -MCFG $G_{\mathbb{B}} = (N, \Gamma, S, P')$ where $\Gamma = \Delta \cup \{\rho^i \mid \rho \in P, i \in [\text{fan-out}(\rho)]\}$ and P' is the smallest set such that for every production $\rho = A \rightarrow [u_1, \dots, u_s](A_1, \dots, A_m) \in P$ there is a production

$$A \rightarrow [\rho^1 u_1, \dots, \rho^s u_s](A_1, \dots, A_m) \in P'. \quad \square$$

Definition 4.2. Let $G = (N, \Delta, S, P, \mu)$ be a non-deleting \mathcal{A} -weighted MCFG. The *weight homomorphism for G* is the \mathcal{A} -weighted alphabetic homomorphism $\text{weights}_G : \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ where $\text{weights}_G(\delta) = 1.\delta$, $\text{weights}_G(\rho^1) = \mu(\rho).\varepsilon$, and $\text{weights}_G(\rho^i) = 1.\varepsilon$ for every $\delta \in \Delta$, $\rho \in P$ and $i \in \{2, \dots, \text{fan-out}(\rho)\}$. \square

$L(G_{\mathbb{B}})$ stands in bijection to D_G via the function *toDeriv* given in Algorithm 7.

Lemma 4.3. $k\text{-MCFL}(\mathcal{A}) = \alpha\text{HOM}(\mathcal{A})(k\text{-MCFL})$

Proof. (\subseteq) Let $L \in k\text{-MCFL}(\mathcal{A})$. By Lemma 2.7 there is a non-deleting \mathcal{A} -weighted k -MCFG $G = (N, \Delta, S, P, \mu)$ such that $\llbracket G \rrbracket = L$. Let f be the function obtained by applying the construction of the rules in $G_{\mathbb{B}}$ position-wise to a derivation in D_G . For every $w \in L(G_{\mathbb{B}})$ we can calculate the corresponding derivation t in G (as a function with domain $\text{dom}(t)$ and labelling function t) using *toDeriv* (Algorithm 7), hence $\text{yield} \circ f$ is bijective. We derive for every $w \in \Delta^*$:

$$\begin{aligned} L(w) &= \llbracket G \rrbracket(w) \\ &= \sum_{d \in D_G(w)} \mu(d) \end{aligned}$$

Algorithm 7 Function *toDeriv* to calculate for every word in $L(G_{\mathbb{B}})$ the corresponding derivation in D_G , cf. Lemma 4.3

Input: $w \in L(G_{\mathbb{B}})$

Output: derivation tree $t \in D_G$ corresponding to w (represented as a partial function from \mathbb{N}^* to P)

```

1: function toDeriv( $w$ )
2:   let  $t$  be the empty function
3:   descend( $t, \varepsilon, 1$ )
4:   return  $t$ 
5: end function

6: procedure descend( $t : \mathbb{N}^* \rightarrow P, \pi \in \mathbb{N}^*, j \in \mathbb{N}$ )
7:   let  $\rho = A \rightarrow [u_1, \dots, u_s](A_1, \dots, A_k) \in P$  and  $u$  such that  $\rho^j u = w$ 
8:   add the assignment  $\pi \mapsto \rho$  to  $t$ 
9:   remove  $\rho^j$  from the beginning of  $w$ 
10:  for every symbol  $\delta'$  in  $u_j$  do
11:    if  $\delta' \in \Delta$  then
12:      remove  $\delta'$  from the beginning of  $w$ 
13:    else
14:      let  $i, j'$  such that  $x_i^{j'} = \delta'$ 
15:      descend( $t, \pi i, j'$ )
16:    end if
17:  end for
18: end procedure

```

$$\begin{aligned}
&= \sum_{d \in D_G} (\text{weights}_G \circ \text{yield} \circ f)(d)(w) && \text{(by } \dagger \text{)} \\
&= \sum_{\substack{d \in D_G, u \in L(G_{\mathbb{B}}) \\ u = (\text{yield} \circ f)(d)}} \text{weights}_G(u)(w) \\
&= \sum_{u \in L(G_{\mathbb{B}})} \text{weights}_G(u)(w) && (L(G_{\mathbb{B}}) \text{ and } D_G \text{ are in bijection)} \\
&= \text{weights}_G(L(G_{\mathbb{B}}))(w)
\end{aligned}$$

For \dagger , one can immediately see from the definitions of f , yield , and weights_G that for every $w \in \Delta^*$ we have $(\text{weights}_G \circ \text{yield} \circ f)(d)(w) = \mu(d)$ if $d \in D_G(w)$ and $(\text{weights}_G \circ \text{yield} \circ f)(d)(w) = 0$ otherwise. Hence $L = \text{weights}_G(L(G_{\mathbb{B}}))$.

(\supseteq) Let $L \in k\text{-MCFL}$ and $h : \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ be an \mathcal{A} -weighted alphabetic homomorphism. By Seki, Matsumura, Fujii, and Kasami [SMFK91, Lemma 2.2] there is a non-deleting k -MCFG $G = (N, \Gamma, S, P)$ such that $L(G) = L$. We construct the \mathcal{A} -weighted k -MCFG $G' = (N, \Delta, S, P', \mu)$ as follows: We extend h to $h' : (\Gamma \cup X)^* \rightarrow ((\Delta \cup X)^* \rightarrow \mathcal{A})$ where $h'(x) = 1.x$ for every $x \in X$ and $h'(\gamma) = h(\gamma)$ for every $\gamma \in \Gamma$. We define P' as the smallest set such that for every $\rho = A \rightarrow [u_1, \dots, u_s](A_1, \dots, A_m) \in P_{(s_1 \dots s_m, s)}$ and $(u'_1, \dots, u'_s) \in \text{supp}(h'(u_1)) \times \dots \times \text{supp}(h'(u_s))$ we have that P' contains the production $\rho' = A \rightarrow [u'_1, \dots, u'_s](A_1, \dots, A_m)$ and $\mu(\rho') = h'(u_1)(u'_1) \cdot \dots \cdot h'(u_s)(u'_s)$. Since \cdot is commutative and G is non-deleting, we obtain $\llbracket G' \rrbracket = h(L(G))$. \blacksquare

By setting $k = 1$ in the above lemma we reobtain the equivalence of 1 and 3 in Theorem 2 of Droste and Vogler [DV13] for the case of complete commutative strong bimonoids.

Example 4.4. Recall the Pr_2 -weighted MCFG G from Example 2.6. By Definitions 4.1 and 4.2 we obtain the MCFG $G_{\mathbb{B}} = (N, \Gamma, S, P')$ where

$$\Gamma = \{a, b, c, d, \rho_1^1, \rho_2^1, \rho_2^2, \rho_3^1, \rho_3^2, \rho_4^1, \rho_4^2, \rho_5^1, \rho_5^2\}$$

and P' is given by

$$\begin{aligned} P' : \quad \rho'_1 &= S \rightarrow [\rho_1^1 x_1^1 x_2^1 x_1^2 x_2^2](A, B) \\ \rho'_2 &= A \rightarrow [\rho_2^1 a x_1^1, \rho_2^2 c x_1^2](A) & \rho'_4 &= A \rightarrow [\rho_4^1, \rho_4^2]() \\ \rho'_3 &= B \rightarrow [\rho_3^1 b x_1^1, \rho_3^2 d x_1^2](B) & \rho'_5 &= B \rightarrow [\rho_5^1, \rho_5^2]() \end{aligned}$$

and the \mathcal{A} -weighted alphabetic homomorphism $\text{weights}_G : \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ where weights_G is given for every $\gamma \in \Gamma$ and $\omega \in \Delta \cup \{\varepsilon\}$ by

$$\text{weights}_G(\gamma)(\omega) = \begin{cases} \mu(\rho_i) & \text{if } \gamma = \rho_i^1 \text{ and } \omega = \varepsilon \text{ for } 1 \leq i \leq 5 \\ 1 & \text{if } \gamma = \rho_i^2 \text{ and } \omega = \varepsilon \text{ for } 2 \leq i \leq 5 \\ 1 & \text{if } \gamma \in \Delta \text{ and } \omega = \gamma \\ 0 & \text{otherwise,} \end{cases}$$

Now consider the (only) derivation $d = \rho_1(\rho_2(\rho_4), \rho_5)$ of $w = ac$. Then

$$\begin{aligned} f(d) &= \rho'_1(\rho'_2(\rho'_4), \rho'_5) & &=: d', \\ g(d') &= \rho_1^1 \rho_2^1 a \rho_4^1 \rho_5^1 \rho_2^2 c \rho_4^2 \rho_5^2 & &=: w', \text{ and} \\ \text{weights}_G(w') &= (1 \cdot 1/2 \cdot 1 \cdot 1/2 \cdot 2/3 \cdot 1 \cdot 1 \cdot 1 \cdot 1) \cdot w & &= (1/6) \cdot w. \quad \square \end{aligned}$$

4.2 Strengthening the unweighted CS representation

Yoshinaka, Kaji, and Seki [YKS10] define (in Section 3.2) an indexed alphabet Δ , a right-linear regular grammar R , and a homomorphism h for some given non-deleting MCFG G that has no rule with at least two identical non-terminals on the right-hand side. We will sometimes write hom_G instead of h to highlight the connection to G . Let S be the initial non-terminal of G . Then R can be viewed as an automaton $\mathcal{M}(G)$ by setting S as the initial state, T as final state, and having for every rule $A \rightarrow wB$ in R (where A and B are non-terminals and w is a terminal string) a transition (A, w, B) in $\mathcal{M}(G)$. For every MCFG G with the above three properties we define the *generator automaton with respect to G* as $\mathcal{M}(G)$ and the *generator language with respect to G* as $R(G) = L(\mathcal{M}(G))$. Note that $\mathcal{M}(G)$ is deterministic. We call $\hat{\Delta}$ the *generator alphabet with respect to G* . By Proposition 3.6 we know that $L(D_{\Delta})$ [YKS10, Definition 1] is a congruence multiple Dyck language, we will denote it by $mD(G)$ to highlight its connection to the MCFG G . For every terminal symbol $\gamma \in \Gamma$, let $\tilde{\gamma}$ abbreviate the string $\llbracket \gamma^{[1]} \rrbracket_{\gamma}^{[1]}$. We give an example to show how the above considerations are used.

Example 4.5 (Examples 4.4 and 2.6 continued). Figure 8 shows the FSA $\mathcal{M}(G')$. An edge labelled with a set L of words denotes a set of transitions each reading a word in L . Note that $R(G')$ is not finite. Let Σ be the generator alphabet with respect to G' . The homomorphism $\text{hom}_{G_{\mathbb{B}}} : \Sigma \rightarrow \Gamma^*$ is given by

$$\text{hom}_{G_{\mathbb{B}}}(\sigma) = \begin{cases} \gamma & \text{if } \sigma = \llbracket \gamma \text{ for some } \gamma \in \Gamma \\ \varepsilon & \text{otherwise.} \end{cases} \quad \square$$

The following is a corollary to Yoshinaka, Kaji, and Seki [YKS10, Theorem 3] where “homomorphism” is replaced by an “alphabetic homomorphism” and “multiple Dyck language” is replaced by “congruence multiple Dyck language”.

Corollary 4.6. Let L be a language and $k \in \mathbb{N}$. Then the following are equivalent:

- (i) $L \in k\text{-MCFL}$
- (ii) there are an alphabetic homomorphism h_2 , a regular language R , and a congruence multiple Dyck language mD_c of at most dimension k with $L = h_2(R \cap mD_c)$.

Proof. The construction of the homomorphism in Yoshinaka, Kaji, and Seki [YKS10, Section 3.2] already satisfies the definition of an alphabetic homomorphism. We may use a congruence multiple Dyck language instead of a multiple Dyck language since, for (i) \Rightarrow (ii), $k\text{-mDYCK} \subseteq k\text{-mDYCK}_c$ (Proposition 3.6) and, for (ii) \Rightarrow (i), $k\text{-mDYCK}_c \subseteq k\text{-MCFL}$ (Lemma 3.7) and $k\text{-MCFL}$ is closed under intersection with regular languages and under homomorphisms [SMFK91, Theorem 3.9]. \blacksquare

Lemma 4.7. For every MCFG G , there is a bijection between D_G and $R(G) \cap mD(G)$.

Proof. The constructions in Lemmas 1 and 3 in Yoshinaka, Kaji, and Seki [YKS10] already hint on the bijection between $R(G) \cap mD(G)$ and D_G , we will merely point out the respective functions $\text{toBrackets} : D_G \rightarrow R(G) \cap mD(G)$ and $\text{fromBrackets} : R(G) \cap mD(G) \rightarrow D_G$ here.

We examine the proof of Lemma 1 in Yoshinaka, Kaji, and Seki [YKS10]. They construct for every rule $A \rightarrow f(B_1, \dots, B_k)$ in G and all tuples $\bar{u}_1, \dots, \bar{u}_k$ that are generated by B_1, \dots, B_k , respectively, a new tuple $\bar{u} = (u_1, \dots, u_m)$ such that $\bar{u} \in L(G_{\Delta})$, for each $i \in [m]$, $\mathcal{M}(G)$ recognises u_i on the way from $A^{[i]}$ to T , and $\text{hom}_G(\bar{u}) = f(\text{hom}_G(\tau_1), \dots, \text{hom}_G(\tau_k))$, where hom_G is applied to tuples component-wise. Now we only look at the initial non-terminal S . Then \bar{u} has only one component and this construction can be conceived as a function $\text{toBrackets} : D_G \rightarrow R(G) \cap mD(G)$ such that $\text{hom}_G \circ \text{toBrackets} = \text{yield}$.

In Lemma 3, Yoshinaka, Kaji, and Seki [YKS10] give a construction for the opposite direction by recursion on the structure of derivations in G_{Δ} . In a similar way as above, we view this construction as a function $\text{fromBrackets} : R(G) \cap mD(G) \rightarrow D_G$ such that $\text{yield} \circ \text{fromBrackets} = \text{hom}_G$. Then $\text{hom}_G \circ \text{toBrackets} \circ \text{fromBrackets} = \text{hom}_G$, and hence $\text{toBrackets} \circ \text{fromBrackets}$ is the identity on $R(G) \cap mD(G)$. \blacksquare

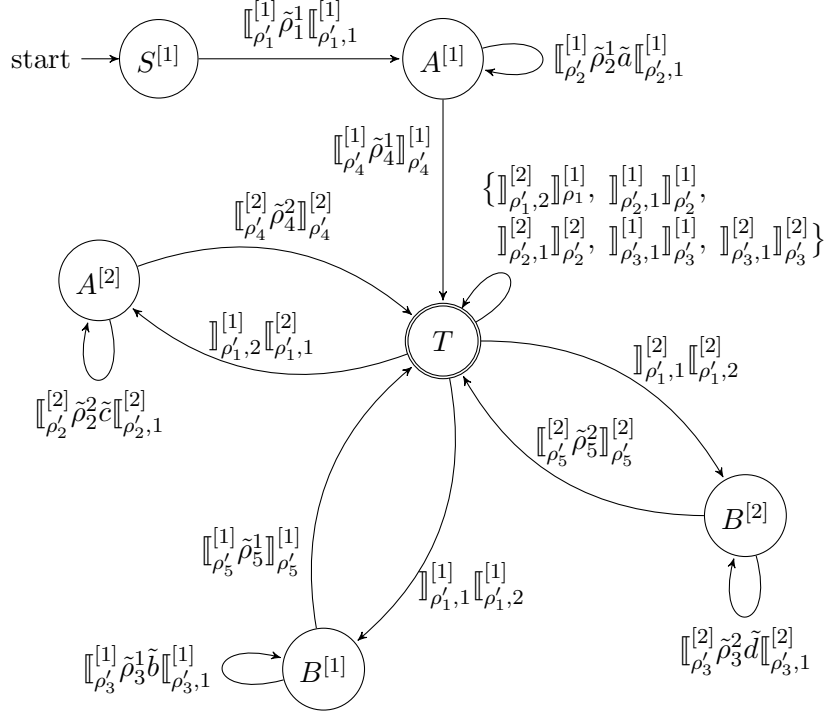


Figure 8: Automaton $\mathcal{M}(G_{\mathbb{B}})$ (cf. Example 4.5)

4.3 Composing the homomorphisms

Lemma 4.8. $\alpha\text{HOM}(\mathcal{A}) \circ \alpha\text{HOM} = \alpha\text{HOM}(\mathcal{A})$

Proof. (\subseteq) Let $h_1 : \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ be an alphabetic \mathcal{A} -weighted homomorphism and $h_2 : \Sigma^* \rightarrow \Gamma^*$ be an alphabetic homomorphism. By the definitions of $\alpha\text{HOM}(\mathcal{A})$ and αHOM there must exist $h'_1 : \Gamma \rightarrow (\Delta \cup \{\varepsilon\} \rightarrow \mathcal{A})$ and $h'_2 : \Sigma \rightarrow \Gamma \cup \{\varepsilon\}$ such that $\widehat{h'_1} = h_1$ and $\widehat{h'_2} = h_2$. Since $h_1(\text{rng}(h'_2)) \subseteq (\Delta \cup \{\varepsilon\} \rightarrow \mathcal{A})$ there is some $h \in \alpha\text{HOM}(\mathcal{A})$ such that $h = h_1 \circ h_2$; hence $h_1 \circ h_2 \in \alpha\text{HOM}(\mathcal{A})$.

(\supseteq) Let $h : \Sigma \rightarrow (\Gamma^* \rightarrow \mathcal{A})$ be an alphabetic \mathcal{A} -weighted homomorphism. Clearly $i : \Sigma^* \rightarrow \Sigma^*$ with $i(w) = w$ for every $w \in \Sigma^*$ is an alphabetic homomorphism. Then we have $h \circ i = h$. \blacksquare

Example 4.9 (Examples 4.4 and 4.5 continued). The homomorphism $h : (\Sigma \cup \bar{\Sigma})^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ obtained from $\text{weights}_G : \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ and $\text{hom}_{G_{\mathbb{B}}} : (\Sigma \cup \bar{\Sigma})^* \rightarrow \Gamma^*$ by the construction for \subseteq in Lemma 4.8 is given for every $\sigma \in \Sigma$ and $\omega \in \Delta \cup \{\varepsilon\}$ by

$$h(\sigma)(\omega) = \begin{cases} \mu(\rho_i) & \text{if } \sigma = \llbracket \rho_i^1 \rrbracket \text{ and } \omega = \varepsilon \text{ for some } i \in [5] \\ 1 & \text{if } \sigma \notin \{\llbracket \rho_i^1 \rrbracket \mid i \in [5]\} \cup \{\llbracket \delta \rrbracket \mid \delta \in \Delta\} \text{ and } \omega = \varepsilon \\ 1 & \text{if } \sigma = \llbracket \delta \rrbracket \text{ and } \omega = \delta \text{ for some } \delta \in \Delta \\ 0 & \text{otherwise.} \end{cases} \quad \square$$

4.4 The weighted CS representation

Theorem 4.10. Let L be an \mathcal{A} -weighted language over Σ and $k \in \mathbb{N}$. The following are equivalent:

- (i) $L \in k\text{-MCFL}(\mathcal{A})$
- (ii) there are an \mathcal{A} -weighted alphabetic homomorphism h , a regular language R , and a congruence multiple Dyck language mD of dimension at most k with $L = h(R \cap mD)$.

Proof. (i) \Rightarrow (ii) There are some $L' \in k\text{-MCFL}$, $h, h_1 \in \alpha\text{HOM}(\mathcal{A})$, $h_2 \in \alpha\text{HOM}$, $mD \in k\text{-mDYCK}_c$, and $R \in \text{REG}$ such that

$$\begin{aligned} L &= h_1(L') && \text{(by Lemma 4.3)} \\ &= h_1(h_2(R \cap mD)) && \text{(by Corollary 4.6)} \\ &= h(R \cap mD) && \text{(by Lemma 4.8)} \end{aligned}$$

(ii) \Rightarrow (i) We use Lemmas 4.3 and 3.7, and the closure of $k\text{-MCFG}$ under intersection with regular languages and application of homomorphisms. \blacksquare

Proposition 4.11. For every \mathcal{A} -weighted MCFG G , there is a bijection between D_G and $R(G_{\mathbb{B}}) \cap mD(G_{\mathbb{B}})$.

Proof. There are bijections between D_G and $L(G_{\mathbb{B}})$ by claims in the proof of Lemma 4.3, between $L(G_{\mathbb{B}})$ and $D_{G_{\mathbb{B}}}$ by claims in the proof of Lemma 4.3, and between $D_{G_{\mathbb{B}}}$ and $R(G_{\mathbb{B}}) \cap mD(G_{\mathbb{B}})$ by Lemma 4.7. \blacksquare

5 Conclusion and outlook

We defined multiple Dyck languages using congruence relations in Definition 3.3, gave an algorithm to decide whether a word is in a given multiple Dyck language in Algorithm 3, and established that multiple Dyck languages with increasing maximal dimension form a hierarchy in Proposition 3.9.

We obtained a weighted version of the CS representation of MCFLs for complete commutative strong bimonoids (Theorem 4.10) by separating the weights from the weighted MCFG and using Yoshinaka, Kaji, and Seki [YKS10, Theorem 3] for the unweighted part.

Theorem 4.10 and Proposition 4.11 may be used to derive a parsing algorithm for weighted multiple context-free grammars in the spirit of Hulden [Hul11].

References

- [CS63] N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. *Computer Programming and Formal Systems, Studies in Logic*, pages 118 – 161, 1963. DOI: 10.1016/S0049-237X(09)70104-1.

- [Den15] T. Denking. A Chomsky-Schützenberger representation for weighted multiple context-free languages. In *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP 2015)*, 2015. URL: <http://aclweb.org/anthology/W15-4803>.
- [DPS79] J. Duske, R. Parchmann, and J. Specht. A homomorphic characterization of indexed languages. *Elektronische Informationsverarbeitung und Kybernetik*, 15(4):187–195, 1979.
- [DSV10] M. Droste, T. Stüber, and H. Vogler. Weighted finite automata over strong bimonoids. *Information Sciences*, 180(1):156–166, 2010. DOI: 10.1016/j.ins.2009.09.003.
- [DV13] M. Droste and H. Vogler. The Chomsky-Schützenberger theorem for quantitative context-free languages. In M.-P. Béal and O. Carton, editors, *Proceedings of the 17th International Conference on Developments in Language Theory (DLT 2013)*, volume 7907 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-38771-5_19.
- [FV15] S. Fratani and E. M. Voundy. Context-free characterization of indexed languages. *CoRR*, abs/1409.6112, 2015. arXiv: 1409.6112.
- [FV16] S. Fratani and E. M. Voundy. Homomorphic characterizations of indexed languages. In A.-H. Dediu, J. Janoušek, C. Martín-Vide, and B. Truthe, editors, *Proceedings of the 10th International Conference on Language and Automata Theory and Applications (LATA 2015)*, pages 359–370. Springer Science + Business Media, 2016. DOI: 10.1007/978-3-319-30000-9_28.
- [HU69] J. E. Hopcroft and J. D. Ullman. *Formal languages and their relation to automata*. Addison-Wesley Longman Publishing Co., Inc., 1969.
- [HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages and computation*. Addison-Wesley, 1st edition, 1979.
- [Hul11] M. Hulden. Parsing CFGs and PCFGs with a Chomsky-Schützenberger representation. In Z. Vetulani, editor, *Human Language Technology. Challenges for Computer Science and Linguistics*, volume 6562 of *Lecture Notes in Computer Science*, pages 151–160. Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-20095-3_14.
- [HV15] L. Herrmann and H. Vogler. A Chomsky-Schützenberger theorem for weighted automata with storage. In A. Maletti, editor, *Proceedings of the 6th International Conference on Algebraic Informatics (CAI 2015)*, volume 9270, pages 90–102. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-23021-4_11.

- [Kal10] L. Kallmeyer. *Parsing beyond context-free grammars*. Springer, 2010. DOI: 10.1007/978-3-642-14846-0.
- [Kan14] M. Kanazawa. Multidimensional trees and a Chomsky-Schützenberger-weir representation theorem for simple context-free tree grammars. *Journal of Logic and Computation*, Jun 2014. DOI: 10.1093/logcom/exu043.
- [Mic01a] J. Michaelis. Derivational minimalism is mildly context-sensitive. In M. Moortgat, editor, *Logical Aspects of Computational Linguistics*, volume 2014 of *Lecture Notes in Computer Science*, pages 179–198. Springer Berlin Heidelberg, 2001. DOI: 10.1007/3-540-45738-0_11.
- [Mic01b] J. Michaelis. Transforming linear context-free rewriting systems into minimalist grammars. In P. Groote, G. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Computer Science*, pages 228–244. Springer Berlin Heidelberg, 2001. DOI: 10.1007/3-540-48199-0_14.
- [Moh00] M. Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1–2):177–201, Mar 2000. DOI: 10.1016/S0304-3975(98)00115-7.
- [Sal73] A. Salomaa. *Formal languages*. Academic Press, 1973.
- [SMFK91] H. Seki, T. Matsumura, M. Fujii, and T. Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229, 1991. DOI: 10.1016/0304-3975(91)90374-B.
- [SNK⁺93] H. Seki, R. Nakanishi, Y. Kaji, S. Ando, and T. Kasami. Parallel multiple context-free grammars, finite-state translation systems, and polynomial-time recognizable subclasses of lexical-functional grammars. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL 1993)*, ACL ’93, pages 130–139, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics. DOI: 10.3115/981574.981592.
- [SS78] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer New York, 1978. DOI: 10.1007/978-1-4612-6264-0.
- [VS87] K. Vijay-Shanker. *A study of tree adjoining grammars*. PhD thesis, 1987.
- [VSWJ86] K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. Tree adjoining and head wrapping. In *Proceedings of the 11th Conference on Computational Linguistics (COLING 1986)*, pages 202–207. Association for Computational Linguistics, 1986. DOI: 10.3115/991365.991425.
- [Wei88] D. J. Weir. *Characterizing mildly context-sensitive grammar formalisms*. PhD thesis, 1988.

- [WJ88] D. J. Weir and A. K. Joshi. Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics (ACL 1988)*, pages 278–285. Association for Computational Linguistics, 1988. DOI: 10.3115/982023.982057.
- [YKS10] R. Yoshinaka, Y. Kaji, and H. Seki. Chomsky-Schützenberger-type characterization of multiple context-free languages. In A.-H. Dediu, H. Fernau, and C. Martín-Vide, editors, *Language and Automata Theory and Applications*, pages 596–607. Springer, 2010. DOI: 10.1007/978-3-642-13089-2_50.